User Manual
of
the MASC Platform

Abstract

MASC (Multi-Agent for Supply Chaining), a Java platform for the management of workshops, is described in the following document. The purpose of this platform is to facilitate the implementation of the approaches imagined by the participants of the French working group COLLINE[1]. It is intended to enable:

- implementing new algorithms,
- defining new scenario for comparing different approaches,
- defining new metrics and comparison criteria.

The platform is composed of three parts:

- the **simulator** which takes a scenario containing all the relative informations on its execution: the sets of containers, operators and stations, as well as all the possible disturbances (breakdowns, strikes etc.).
- the **scheduler** being used as a communication interface between the simulator and the solving system developed by the user. It provides the data of the studied case to the solver and retrieves the solution. It is developed by the user.
- the **solver**, also developed by the user, is the central point of the platform. It takes the data provided by the scheduler, looks for a solution to the submitted problem and sends back a list of assignments to the simulator by the means of the scheduler.

1 www.irit.fr/COLLINE

Table of Contents

1.Role of the platform....4
1.1.Management of the Simulation Window....4
1.1.1.Access to the Scenario....4
1.1.2.Simulation Control....4
1.1.3.Languages....4
1.2.Management of the Scenarios Window....4
1.2.1.Creation of a Scenario....4
1.2.2.Validity Check....4
1.2.3. Display....5
2.Operation Principle....5
2.1.Principle of the Simulator....5
2.2.Scheduling Principle....5
2.3.Activity of an Entity....6
2.4.Managing Disturbances....6
2.5.Machinings or Qualifications....6
2.6.Validity Check of Planed Actions....7
3.Interfaces of MASC....7
3.1.Main Window....7
3.1.1.Menu Bar....8
3.1.1.1.Scenario Management....8
3.1.1.2.Scenario Execution Control....8
3.1.1.3.How to Choose a Language....9
3.1.2.Task Bar....9
3.1.3.Factory Panel....9
3.1.3.1.Containers....10
3.1.3.2.Operators....10
3.1.3.3.Stations....10
3.1.3.4.Container Machining....11
3.1.3.5.Characteristics of Entities....11
3.1.4.Properties Panel....11
3.1.5.Results Graph Panel....12

3.1.6.Trace Panel....13
3.2.Scenario Creation Window....13
3.2.1.Window menu....14
3.2.2.Manual Creation of a Scenario....15
3.2.3.Automatic Creation of a Scenario....16
3.2.4.Scenario Error Window....17
4.Platform....17
4.1.Masc.zip File....17
4.2.Parts to Develop....18
5.Development Environment....19
6.Annexe....19
6.1.Example of a Scheduler: BasicScheduler....19
6.2.Classes Diagram for the Scheduler....22
6.3.Classes Diagram for Entities....23

THE MASC PLATFORM

1. ROLE OF THE PLATFORM

The provided environment of development must help each participant of the Colline working group for the implementation of his approach. The first interface enables launching a simulation, the second one creating scenarios. This section specifies the functionalities of these tools. A more detailed description is provided in the following sections of this document.

Throughout this document, a station, an operator or a container will equally be called “entity”.

1.1. Management of the Simulation Window

The window appearing when a simulation is launched concentrates all the functionalities of the development tools. It is composed of three menus.

1.1.1. Access to the Scenario

This menu enables creating, displaying or executing a scenario.

1.1.2. Simulation Control

This control may be done according to two modes:

- A “Step by step” one in which a stop takes place after each cycle of the simulator,
- A “Continuous” one in which a stop takes place at the end of the simulation or when the users clicks on “stop”.

This menu enables also stopping, making a pause or starting again a session.

1.1.3. Languages

The user may chose between the French or English texts inside the windows.

1.2. Management of the Scenarios Window

This window enables building up a scenario by specifying the characteristics of its various elements.

1.2.1. Creation of a Scenario

A scenario contains the description of:

- Entities (which may be stations, operators or containers),
- Required machinings or qualifications,
- Disturbances which may occur during the simulation.

A scenario can be created manually or automatically as described thereafter.

1.2.2. Validity Check

The control executed on the data enables to detect inconsistencies in the scenario:

- A scenario must contain at least one station, one operator and one container,
- Each entity must have at least one qualification,
- Identifiers of entities must be unique,
- All the qualifications of containers must be processed by the operators and the stations of the scenario.

1.2.3. Display

The scenario is created using a structure in a tree form. A display function enables to visualize it in a textual form.

2. OPERATION PRINCIPLE

2.1. Principle of the Simulator

The platform manages the events occurring during a simulation. Different types of events are identified:

- Disturbances defined by the scenario; they may occur on a station or an operator,
- Availabilities of entities,
- Moves of operators and containers. Some moves are defined in the scenario, such as the arrival of containers in the factory. Two other moves are determined by the scheduler: move of one container and one operator on a station at a cycle C,
- Machining management. A container contains a definite number of pieces, each piece requiring an ordered machining sequence. The simulator controlling that machining can be executed by the operators on the stations (as well as its duration) can be executed by the operators on the station. A container is processed and may leave the factory when it does not contain anymore machining. The execution time of a machining is unique whatever the station or the operator executing this machining.

The simulator also manages the messages to be displayed in the different windows of the application.

2.2. Scheduling Principle

During each simulation cycle, the scheduler determines moves of the entities, starting from a pool of available entities defined by the simulator and answering the three following criteria:

- The entity is present in the factory,
- The station or operator did not receive a disturbance,
- The entity is inactive.

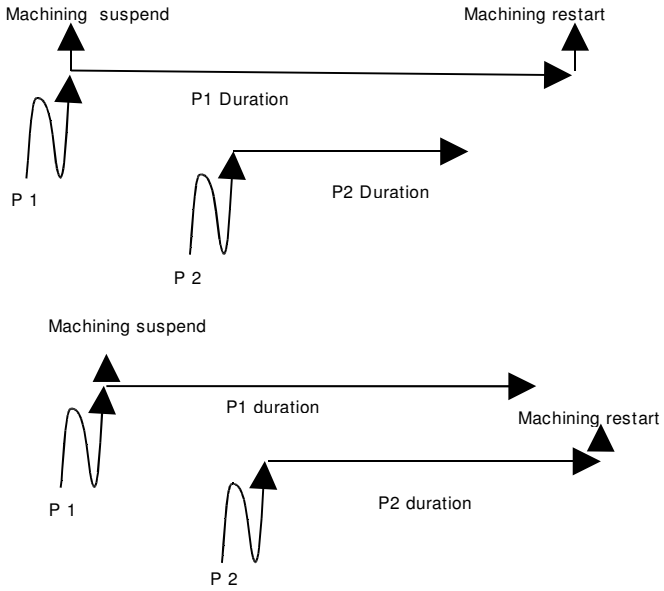
These moves are presented in the form of a pair <station, operator> or <station, container> that will be called “moving”. Such a couple means that the station will receive the operator or the container during the current cycle. When the station has an operator and a container, it is able to execute the first machining on the pieces of the container.

2.3. Activity of an Entity

The activity duration is defined by the formula: MachiningDuration = F (number of pieces contained in the container * processing time of the first machining).
A move is correct if and only if the operator and the station are able to execute the first machining defined by the container. In the same way, an operator or a container can move on a station only if this latter is free. When the entity (container or operator) positioned on the station has finished the first machining of the container, it leaves the station and joins the list of available entities.

2.4. Managing Disturbances

The disturbances regard stations or operators. A station which has received a disturbance is known as “broken down”, whereas an operator in the same case is known as “inalienable”. If the disturbance concerns a station or an active operator, the machining duration of the container is increased by the duration of the disturbance. The container and the operator remain on standby on the station until the end of the disturbance. A disturbance on an entity can intervene while the entity is already broken down or inalienable, in this case the entity will be again available at the end of the longest disturbance.



2.5. Machinings or Qualifications

Each container has a scheduled sequence of machining. The container will leave the factory when its list of machining is empty. The simulator manages this list of machining during the whole simulation. Thus a container present in the list of the available containers will have only machinings to be executed. In the simulator, such as we built it, the containers remain on the broken down stations. A disturbance of infinite duration will block the processing of a container and simulation will not be able to finish. In the same way, when an involved operator at a station becomes inalienable, it blocks this station with the container in the course of processing, until the end of the disturbance.

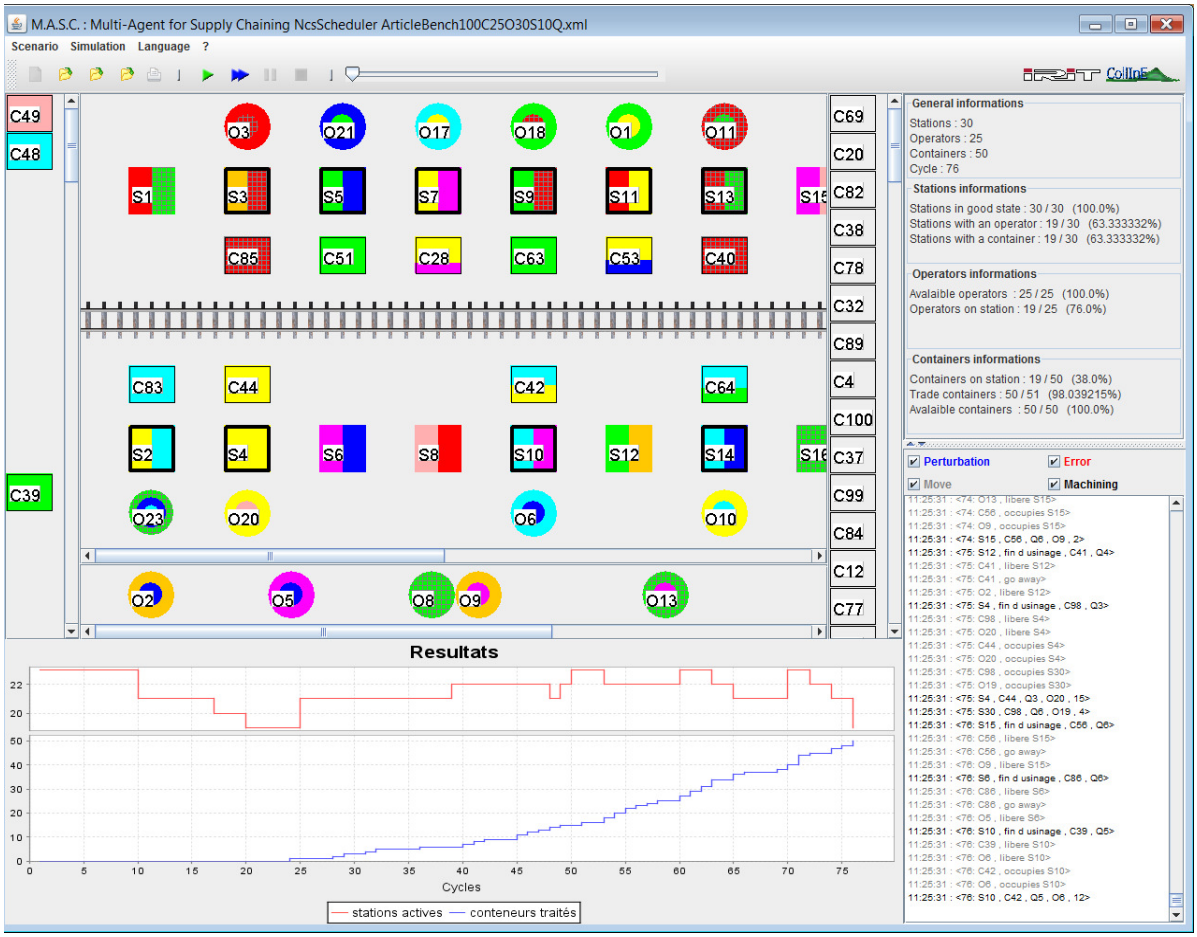
2.6. Validity Check of Planed Actions

Controls are executed by the simulator on the assignments detected by the scheduler:

- Error of occupancy: the operator or the container is already on a station, the station is not free,
- Error of qualification: the operator or the station does not have the qualification for processing the first machining of the container,
- Error of uselessness: the operator, the container or the station do not have a qualification.

3. INTERFACES OF MASC

3.1. Main Window

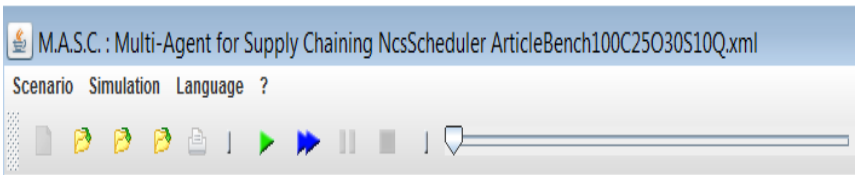

M.A.S.C. : Multi-Agent for Supply Chaining NcsScheduler ArticleBench100C25O30S10Q.xml
Scenario Simulation Language ?
General informations
Stations : 30
Operators : 25
Containers : 50
Cycle : 76
Stations informations
Stations in good state : 30 / 30 (100.0%)
Stations with an operator : 19 / 30 (63.333332%)
Stations with a container : 19 / 30 (63.333332%)
Operators informations
Avalaible operators : 25 / 25 (100.0%)
Operators on station : 19 / 25 (76.0%)
Containers informations
Containers on station : 19 / 50 (38.0%)
Trade containers : 50 / 51 (98.039215%)
Avalaible containers : 50 / 50 (100.0%)
Perturbation
Error
Move
Machining
Resultats
Cycles
stations actives
conteneurs traités

The interface has a menu bar, a task bar and is divided into four panels:

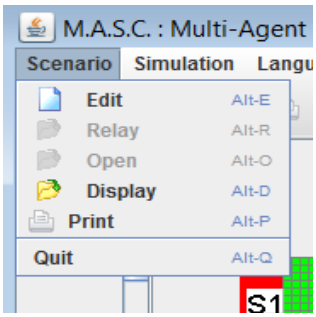
- The factory panel,
- The graph panel,
- The information panel,
- The trace panel.

3.1.1. Menu Bar

The title of the main window displays the name of the scheduler and scenario that are used.



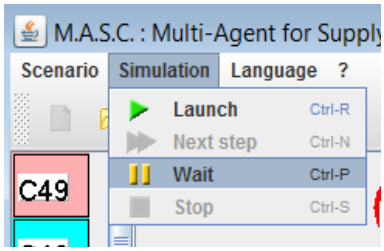
3.1.1.1. Scenario Management



The "Scenario" menu gives several items:

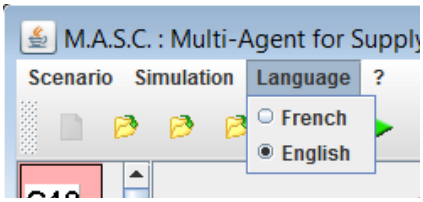
- **Edit**: this enables editing a scenario.
- **Replay**: this replays a scenario, starting from the trace file (see panel of trace) of a previous simulation. These trace files are the textual files recorded in the "res" directory. A file, with an "xml" extension, may be find in the directory "data".
- **Open**: this enables opening a scenario in order to execute a session.
- **Display**: displays a scenario on a screen.
- **Print**: this enables scenario printing.

3.1.1.2. Scenario Execution Control



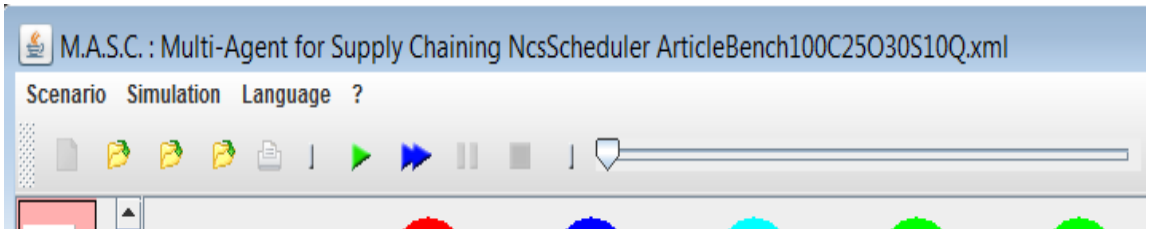
The control is defined by the menu items in the continuous mode (Launch) or the step-by-step one (NextStep) In the step-by-step mode, several cycles may occur before a stop is effective, because simulation is stopped only when one action (moves of entities, breakdown, machining...) is totally achieved.

3.1.1.3. How to Choose a Language



This may be done using the “Language” item of the “Menu” bar. The user may choose between French and English.

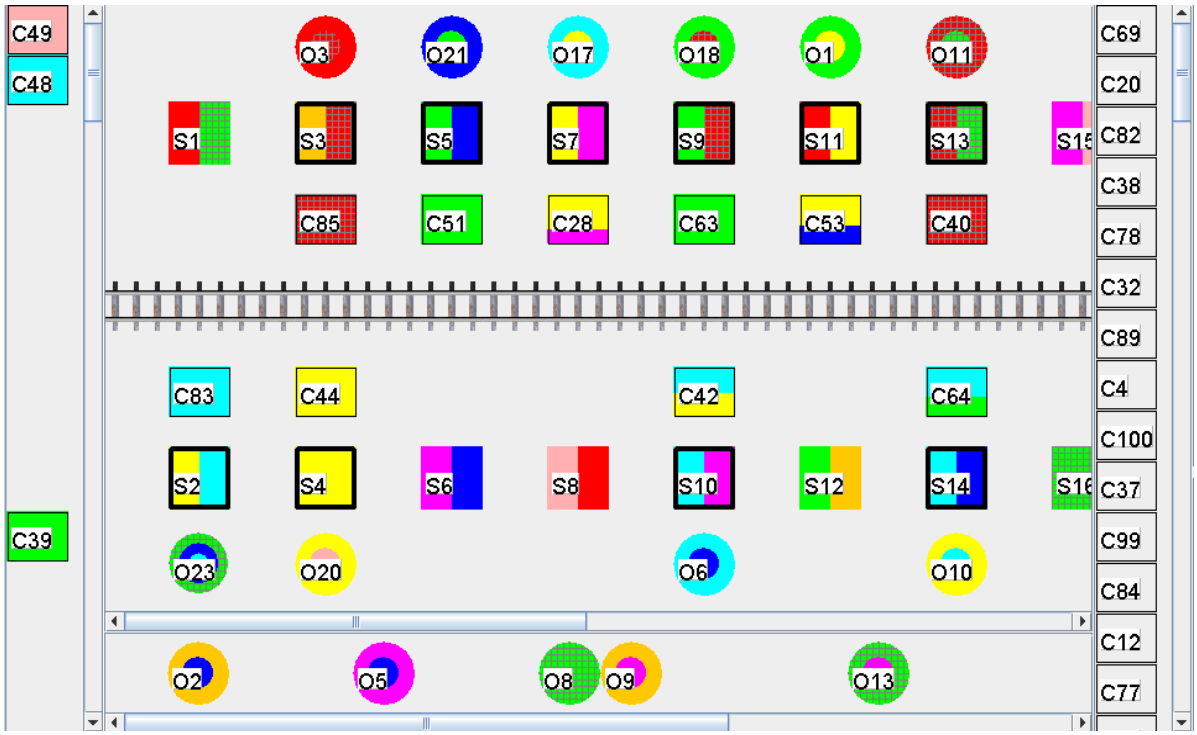
3.1.2. Task Bar



This bar takes again the functionalities of the menu bar, the cursor makes it possible to modulate the execution speed of the simulation.

3.1.3. Factory Panel

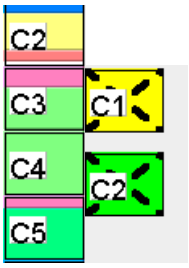
This panel enables visualizing the moves of the operators and the containers on the stations.



This panel is divided into four areas:

- The input area of the containers, it contains the containers present in the factory and that are not allocated to a station,
- The staff room in which operators are waiting for their next appointment,
- The storage area of the machined containers, in this window are stored the containers for which all the pieces are processed,
- The workshop where machinings are executed. At the beginning of simulation, this area contains only the stations, distributed on both sides of a provisioning rail. The operators assigned to the station as well as the containers in progress are then dynamically represented.

3.1.3.1. Containers

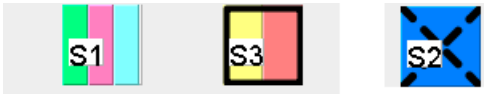


The containers are represented by rectangular boxes. The different colours represent the qualifications necessary for their machining; the higher colour represents the next machining to be executed. The size of the various coloured zones depends on the qualification duration and the number of pieces stored in the container. Machinings disappear as they are finished. A white container means that all its machinings are executed. A container with a cross in the input area means that it could not be processed in the time which was allocated to it.

3.1.3.2. Operators

Operators are represented by circle shapes. Concentric colours represent the qualifications owned by the operator. A non available operator is tagged with a black cross.

3.1.3.3. Stations



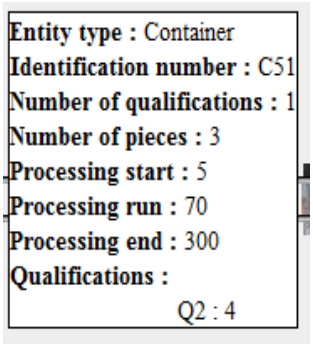
The three existing types of a station are represented here. Machinings being able to be executed by a station are represented by vertical bars of different colours. An active station is surrounded by a black edge. In the same way that an operator, a broken down station is marked with a black cross.

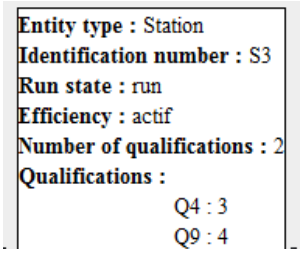
3.1.3.4. Container Machining

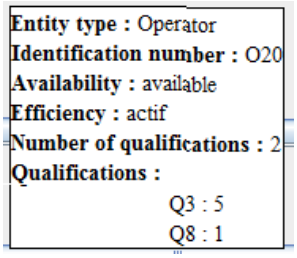
When a machining is processed by a station, the three involved entities are displayed on the same column. In the upper display we can visualize that the qualification necessary for the machining of the container (in this case, of pink colour) exists at the same time on the station and on the operator.

3.1.3.5. Characteristics of Entities

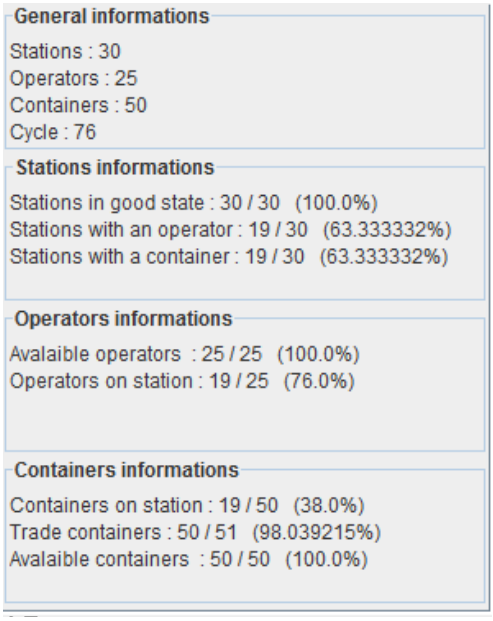
They are displayed by two clicks with the left button on the corresponding icon.







3.1.4. Properties Panel


General informations
Stations : 30
Operators : 25
Containers : 50
Cycle : 76
Stations informations
Stations in good state : 30 / 30 (100.0%)
Stations with an operator : 19 / 30 (63.333332%)
Stations with a container : 19 / 30 (63.333332%)
Operators informations
Avalaible operators : 25 / 25 (100.0%)
Operators on station : 19 / 25 (76.0%)
Containers informations
Containers on station : 19 / 50 (38.0%)
Trade containers : 50 / 51 (98.039215%)
Avalaible containers : 50 / 50 (100.0%)

Some details are given on this panel.

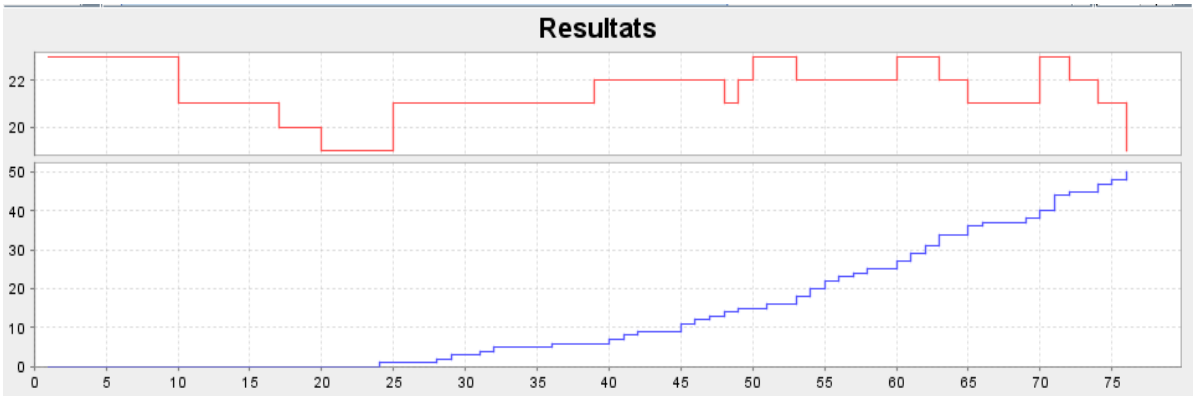
• **General information**: the entities that are present in the factory during the given cycle are listed here.

• **Information about the containers**:

– Containers on a station: This area lists the containers which are present in the factory. Containers may arrive during the simulation and they are not counted anymore when their processing is achieved.

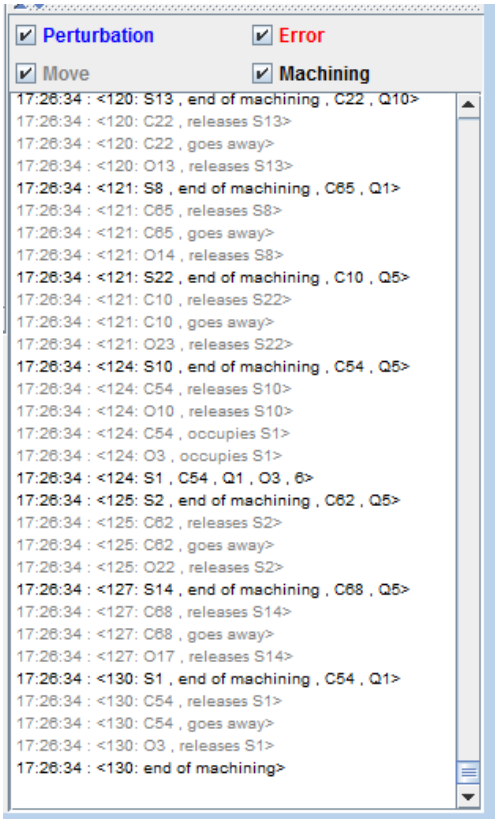
– Processed containers: this gives all the containers, the simulation is finished when 100% of the containers were processed.

3.1.5. Results Graph Panel



This window enables displaying the processed containers, as well as the active stations, using curves in a graph.

3.1.6. Trace Panel



This panel displays the actions executed by the entities followed by the number of the starting cycle. When all the containers are processed, a message of end of processing is displayed in this window.

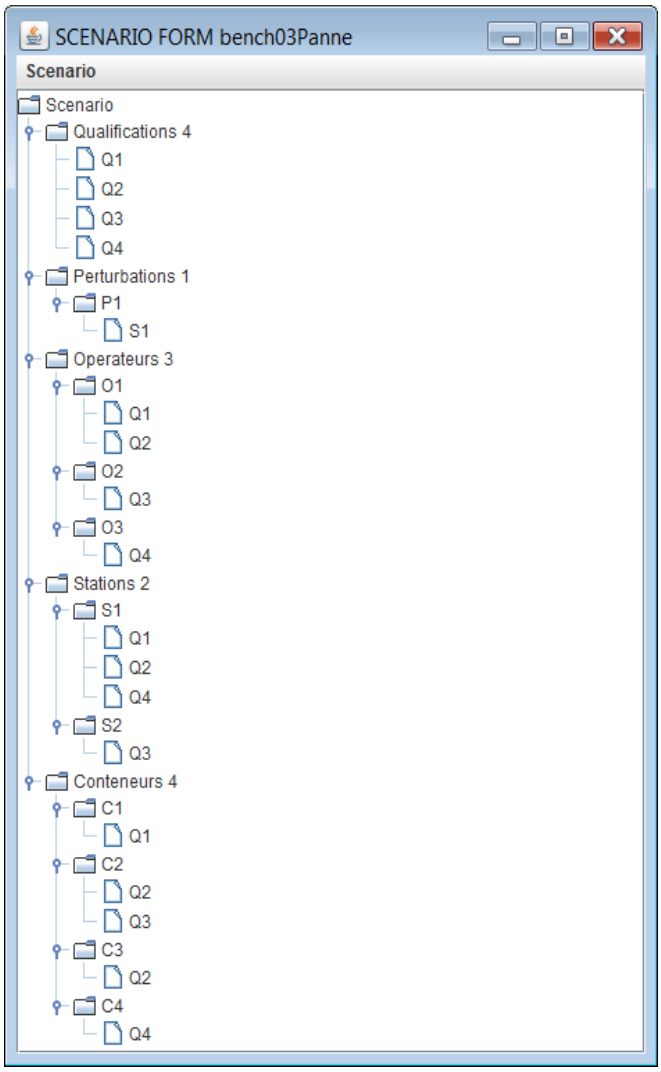
Different kinds of displays may be selected:

- Disturbances, in blue,
- Moves, in grey,
- Errors of assignment, in red,
- Machining, in black.

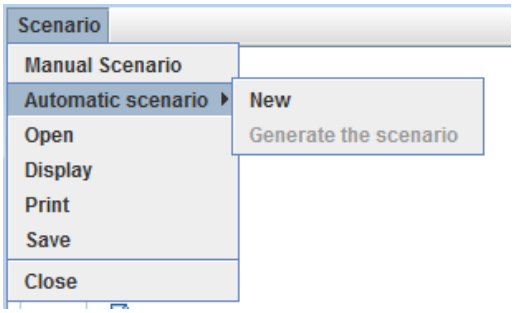
The trace of this execution is also safeguarded in a file called “name of the scenario”. “txt”.

3.2. Scenario Creation Window

This window enables to create the different elements of a scenario by handling a tree.



3.2.1. Window menu



This menu respectively enables:

- Creating a new scenario manually,
- Creating a scenario automatically,
- Opening an existing scenario,

- Displaying a scenario,
- Print a scenario,
- Saving a scenario in an “xml” file,
- Closing this window.

3.2.2. Manual Creation of a Scenario

By clicking once with the right button, the different elements of a scenario may be created.

By clicking on the node corresponding to the type of the element to create, a new entity in the scenario is created. The identifiers of the created element are automatically generated.

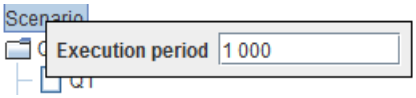
By clicking on an element identifier, one associates respectively:

- Stations and operators with disturbances,
- Qualifications with the stations, operators and containers.

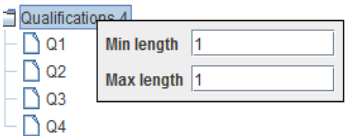
By clicking twice with the right button, an element of a scenario is removed. Update of the entities involved is automatically executed (qualification of the entities, entities related to a disturbance).

Clicking once with the left button on a node modifies the characteristics of this node. The default values are displayed in number of cycles. A cycle is the elementary unit value allowing the processing of various machinings.

➢ Characteristics of a scenario

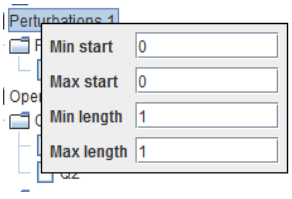


➢ Characteristics of qualifications



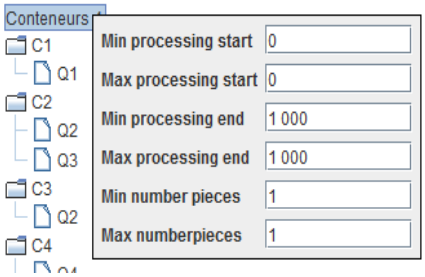
The duration of a generated qualification will be included between the bounds.

➢ Characteristics of disruptions



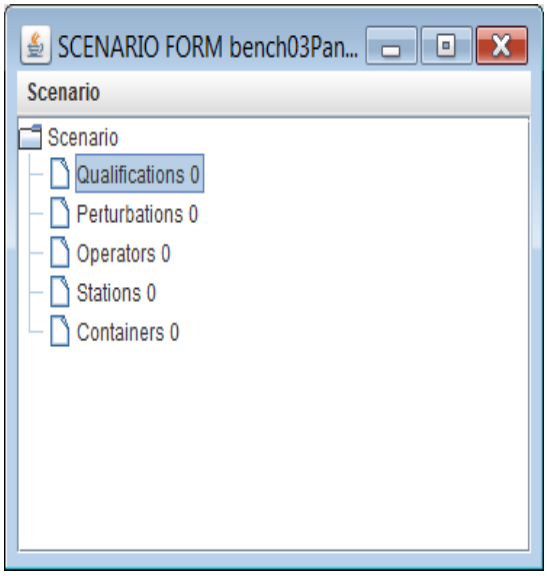
The beginning and the duration of a generated disturbance will be included between the bounds.

- Characteristics of containers



The beginning and the ending processing of a container, as well as the number of pieces of this container, have to be included between the min and max bounds. The completion date of processing is by default the completion date of the scenario.

3.2.3. Automatic Creation of a Scenario

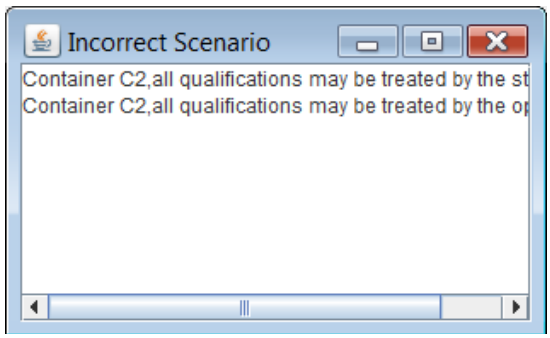


In addition to the manual characteristics of a scenario, one must specify:

- The number of elements to be generated,
- The minimum and maximum number of entities concerned with each disturbance,
- The minimum and maximum number of qualifications for each entity.

Generating the scenario creates the entities, their qualifications and their disturbances.

3.2.4. Scenario Error Window


Incorrect Scenario
Container C2,all qualifications may be treated by the st
Container C2,all qualifications may be treated by the op

When the scenario is saved, the completeness of this scenario is tested:

- A scenario must comprise at least one station, one operator and one container,
- All the qualifications of all the containers must be processed by at least one station and one operator,
- The identifiers of the entities must be unique.

4. PLATFORM

The platform is provided[2] in the form of a compressed file called *masc.zip*. It was developed according to the View-Controller model.

4.1. Masc.zip File

Once decompressed in a directory specific to the platform, this file contains the following directories and files:

- **src:** sources of the files:
  - Controler: represents the application behaviour,
  - Interface: contains the interface code with which the user interacts.
- **Images:** contains the icons used.
- **Languages**: contains the files necessary for the internationalization of the interface i.e. the texts to be displayed in English and French.
- **data**: contains the scenarios files of the application. These files have an “xml” extension. They were generated at exit of the window of scenario creation. This directory also contains the **s***cheduler.txt* file. This file must be modified when the behaviour of the agents and of the scheduler are implemented. It contains the path where the Java abstract machine will seek the code of the developed scheduler.
- **doc**: the javadoc of the used classes.
- **lib**: the files *jfreechart-1.0.2.jar* and *jcommon-1.0.5.jar* contain the libraries necessary for displaying curves.

2 www.irit.fr/COLLINE

• **res**: ".txt" textual files contain the log of the actions (moves, machinings, disturbances…) executed by the entities.

This file also contains the *masc.jar* executable file.

4.2. Parts to Develop

The developer has to create its own scheduler. This class implements the interface

• Scheduler doc\controler\generic\simulation\Scheduler.html. It is necessary to define the five following methods:

- buildConteneur()
- buildOperateur()
- buildStation()
- buildQualification()
- executeStep(available stations vector, available containers vector, available operators vector, active station vector, number of cycle)

The scheduler is called only when "something" appends in the platform (disruption, end of processing of a container...), it returns the vector of the assignments to be realized for this simulation cycle, cf. Affectation doc\controler\generic\simulation\Affectation.html

- setOperateur(@operateur)
- setConteneur(@ conteneur)
- setStation(@station)

It has also to describe the methods defining the behavior of the entities used in the platform.

Each entity inherits respectively one of the following classes:

Container doc\controler\generic\factory\Conteneur.html

- getContent() number of pieces in the container
- getStation() station where the container is assigned
- getCurrentQualification() qualification currently treated
- getQualifications() qualifications not yet treated
- getIdentificateur() identifier of the container

Station doc\controler\generic\factory\Station.html

- getOperateur() operator being on the station
- getConteneur() container being on the station
- getQualifications() vector of the qualifications which the station can execute
- getIdentificateur() identifier of the station

Operator doc\controler\generic\factory\Operateur.html

- getStation() station where the operator is assigned
- getQualifications() vector of the qualifications the operator can execute
- getIdentificateur() identifier of the operator

These various entities have by default the methods to update the MMI. They inherit the Entite class defining the methods getQualifications () and getIdentificateur ().

Entity doc\controler\generic\factory\Entite.html

The developer has also information about the qualifications. He can attribute his own characteristics to them; in this case, he will define his own class "qualification" inheriting the Qualification class.

Qualification doc\controler\generic\factory\Qualification.html

- getTempsExecution() processing time of a qualification for a piece.
- getIdent()identifier of the qualification.

We commented only the methods useful for the developer.

5. DEVELOPMENT ENVIRONMENT

Version 1.5 or later of Java should be used.

If an error of the JVM occurs blaming the *libfontmanager.so*, it is about a problem of AWT which does not accept certain fonts (a priori it is the case for the Debian distribution and derived ones such as Ubuntu etc.). In general, Rekha-normal and aakar-MagNet are in question. To solve the problem, remove the corresponding font by writing the command: *sudo apt-get remove ttf-gujarati-fonts*.

This can make certain meta-packages orphan but that does not have any incidence.

If an error of the type:

```
Exception in thread "main" java.lang.NullPointerException
at javax.swing.ImageIcon.<init>(Unknown Source)
```

occurs at execution time, modify the *classpath* with the root of the project.

6. ANNEXE

6.1. Example of a Scheduler: BasicScheduler

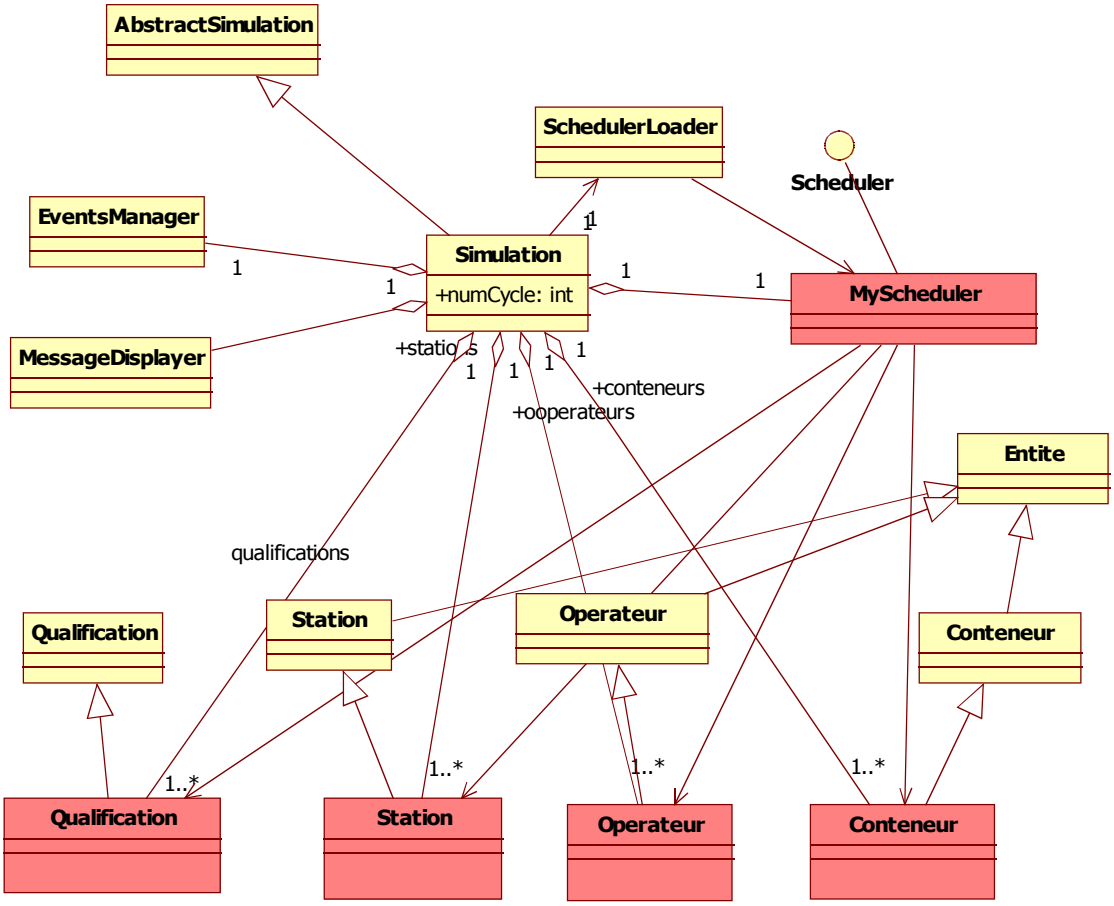
```
package controler.basic.simulation;

import java.util.Enumeration;
import java.util.Vector;

import controler.generic.factory.Entite;
import controler.generic.factory.Conteneur;
```

```
import controler.generic.factory.Operateur;
import controler.generic.factory.Qualification;
import controler.generic.factory.Station;
import controler.generic.simulation.*;

/**
 * @author machonin
 * */
class BasicVector     extends Vector     {
        public BasicVector(Vector _vector)
        {
                for (Enumeration en2 =  _vector.elements(); en2.hasMoreElements();)
                {
                        Entite element = (Entite) en2.nextElement();
                                add(element);
                }
        }

        Entite find (Conteneur conteneur)
        {
                for (Enumeration en2 = elements(); en2.hasMoreElements();)
                {
                        Entite element = (Entite) en2.nextElement();
                        if
(element.getQualifications().contains((Qualification)conteneur.getQualifications().elementAt(0)))
                        {
                                return element;
                        }
                }
                return null;
        }
}

public class BasicScheduler  implements Scheduler <Station, Conteneur, Operateur>
{
        public BasicScheduler()
        {
                super();
        }

        public Conteneur  buildContainer ()
        {
         return new Conteneur();
        }

        public Operateur  buildOperateur ()
        {
```

```
		return new Operateur();
	}

	public Station  buildStation ()
	{
		return new Station();
	}

	public Qualification  buildQualification ()
	{
		return new Qualification();
	}

	public BasicVector conteneurs = null;
	public BasicVector stations = null;
	public BasicVector operateurs = null;

	public  Vector<Affectation> executeStep(Vector<Station> _stations, Vector<Conteneur>
_conteneurs, Vector <Operateur> _operateurs, Vector <Stations> _stationsActives, final int
_numCycle)
	{
		Vector <Affectation> affectations =  new Vector<Affectation>();

		stations = new BasicVector(_stations);
		operateurs = new BasicVector(_operateurs);
		conteneurs = new BasicVector(_conteneurs);
		Conteneur conteneur = null;
		Station station = null;
		Operateur operateur = null;

		while (!conteneurs.isEmpty())
		{
			conteneur = (Conteneur)conteneurs.firstElement();
			conteneurs.remove(conteneur);
			station = (Station) stations.find (conteneur);
			if (station != null)
			{
				operateur = (Operateur) operateurs.find (conteneur);
				if(operateur != null)
				{
					stations.remove(station);
					operateurs.remove(operateur);
					Affectation affectation = new Affectation();
					affectation.setStation(station);
					affectation.setOperateur(operateur);
					affectation.setConteneur(conteneur);
					affectations.add(affectation);
```

```
                    }
                }
            }

            return affectations;
        }
    }
```

6.2. Classes Diagram for the Scheduler



Classes which have to be develop are coloured in red.

Provided classes are coloured in yellow.

6.3. Classes Diagram for Entities